# IMAGE: A DESIGN INTEGRATION FRAMEWORK APPLIED TO THE HIGH SPEED CIVIL TRANSPORT

**Mark A. Hale**
*Graduate Researcher - Aerospace Systems Design Laboratory*
*GSRP Fellow, NASA Langley HPCC*

**James I. Craig**
*Professor - School of Aerospace Engineering*
*Co-Director - Aerospace Systems Design Laboratory*

*Georgia Institute of Technology*
*School of Aerospace Engineering*
*Atlanta, Georgia 30332*

## ABSTRACT

Effective design of the High Speed Civil Transport requires the systematic application of design resources throughout a product's life-cycle. Information obtained from the use of these resources is used for the decision-making processes of Concurrent Engineering. Integrated computing environments facilitate the acquisition, organization, and use of required information. State-of-the-art computing technologies provide the basis for the Intelligent Multi-disciplinary Aircraft Generation Environment (IMAGE) described in this paper. IMAGE builds upon existing agent technologies by adding a new component called a model. With the addition of a model, the agent can provide accountable resource utilization in the presence of increasing design fidelity. The development of a zeroth-order agent is used to illustrate agent fundamentals. Using a CATIA™-based agent from previous work, a High Speed Civil Transport visualization system linking CATIA, FLOPS, and ASTROS will be shown. These examples illustrate the important role of the agent technologies used to implement IMAGE, and together they demonstrate that IMAGE can provide an integrated computing environment for the design of the High Speed Civil Transport.

**KEYWORDS:** IMAGE, computer, integration, environment, framework, agent, wrapping, resources, networking, CATIA

## BACKGROUND

Concurrent Engineering (CE) formalizes a concurrent decision-making process, as shown in Figure 1.[1] Product and process driven engineering tasks provide information while decision-support methods are used to make decisions. A computing environment facilitates the acquisition, organization, and application of information and integrates the engineering processes.

A number of pilot projects have been implemented that investigate integrated computing issues. These projects include integrated design frameworks (FIDO [2], HiSAIR/Pathfinder [3,4], PACT [5], Designworld [6], Prism [7]), modular conceptual design systems (ACSYNT [8], FLOPS [9]), and quasi-procedural systems (PASS [10,11]). At the Aerospace Systems Design Laboratory (ASDL), two design frameworks are being developed to investigate life-

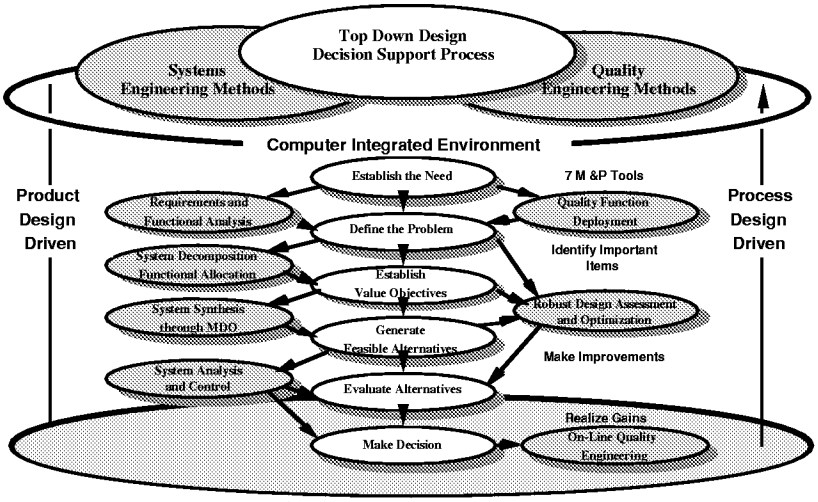


**Figure 1.** Concurrent Engineering

cycle design. LEGEND (Laboratory Environment for the Generation, Evaluation, and Navigation of Design) is a prototype system for quantifying, developing, and instantiating designs.[12] IMAGE (Intelligent Multi-disciplinary Aircraft Generation Environment), the subject of this paper, is currently under development by the authors.[13] IMAGE addresses two fundamental issues: formulation of a design model and development of enabling computational technologies to implement a design environment. The latter is the focus of this paper.

Research has shown that one of the key technologies necessary for implementation of an integrated computing environment is the *agent*. Building on existing agent definitions, IMAGE adds a new component: the *model*. With a model, agents can be used flexibly to generate design information and can be held accountable for their actions. This paper will define key characteristics of these extended agents and illustrate the benefits of using agents in integrated computing environments.

## AGENT DEFINITION

The following agent definition is proposed by the authors:

> ***An agent is a resource that has been modeled and wrapped for inclusion in a distributed design environment. Agent design requires a designer-centered, bi-directional wrap that is independent of proprietary boundaries and capable of supporting increasing fidelity models.***

This definition characterizes an agent by its components and behavior. There are three agent components: the resource, the model, and the wrap. Agents must accommodate information obtained from heterogeneous resources and must apply this to design models of increasing fidelity across a product's life-cycle. Agents are one of the key integration tools for a distributed, designer-centered, multi-tasking design environment.

For the first time the role of proprietary resources and information is explicitly stated in the agent definition. Proprietary resources are generally stand-alone in nature, with limited communications capabilities, and preserve software rights through a number of advanced computing techniques. Together, these present a formidable challenge in the development of integrated design environments. Finally, it should be noted that proprietary information must be accommodated and secured in open, integrated environments.

## AGENT COMPONENTS

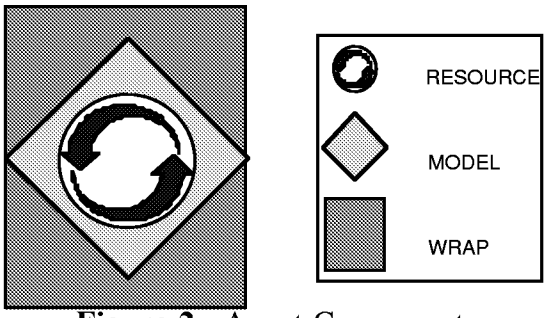

**Figure 2.** Agent Components

As defined in IMAGE, a generalized agent is either a *tool* or an *agent*. Both incorporate resources and are used to produce design information or make design decisions. A *tool* is the most basic type of generalized agent and is comprised of a resource, typically a computer program, and a wrap, typically program utilities used for communicating with other tools and agents. An *agent,* as shown in Figure 2, is a tool that, along with a resource and a wrap, also includes a model. With the addition of the model, the agent can generate *accountable* design information to be used for making decisions. Accountability was first introduced in LEGEND[13], and accountable information is defined in IMAGE as information with the context in which it was developed. Context, in this situation, includes the "what, why, when, and how" information attributes on which accountability can be based.

Agents and tools are the basic elements used in IMAGE to implement an integrated computing environment. Agents operate on the basis of the models that they contain and therefore can provide accountability for the information they produce. On the other hand, tools which do not include a model can produce only information with no context and therefore no inherent accountability. Tools can be as equally useful as agents, but they must be used by either other agents or design experts, either of which must provide the appropriate accountability.

### Model

The model adds context to the information produced by an agent and, therefore, provides accountability. Models are typically based on mathematical formulations, engineering principles, or geometrical constructions. In addition, models may also include limitations, units, and details of implementation of the agent.

### Resource

Resources are entities that produce additional design information based on existing information. Typical resources include off-the-shelf computer programs such as ASTROS (a structural optimization code), FLOPS (an aircraft convergence code), ACSYNT (an aircraft convergence code), CONMIN (an optimization package), CATIA (a three-dimensional geometric modeling, simulation and analysis package), and ORACLE (a relational database). Often overlooked, the design expert (the designer) and design experience are also design resources. Knowledge-Based Systems can be used to capture design expert knowledge, while "lessons learned" can be captured in experience-related resources.

### Wrap

The wrap manages information generation within an agent and transfer between agents. The wrap implements bi-directional information exchange within the design environment. For computer-based resources, the communication channel needs to be accessible through the multi-user, multi-platform, multi-language, networked workstation systems used in current design systems. A tool that has been successfully used for inter-agent communications in IMAGE is the Tk/tcl utility package developed at U.C.-Berkeley.[14] [Note: Tk/tcl is an interpretive, X11 windowing system.]

As mentioned earlier, the accessibility of design resources varies significantly between proprietary and non-proprietary codes. Nonproprietary codes are often easier to wrap because source code level access is available. Therefore, wrapping utilities can be directly integrated by

restructuring the source code itself. In contrast, proprietary resources are usually provided in an object/executable form. Fortunately, internal resources of mature commercial software products can often be integrated with link-edit procedures, and this can form the basis for agent wrapping. The ability to wrap proprietary resources is important because it presents a considerable step toward design automation.[15]

## ZEROTH-ORDER AGENT

The zeroth-order agent, or tool, is characterized by two features: the model is not explicitly defined in the agent but rather is implicitly defined in the resource, and the input/output stream of the resource is wrapped. Since the model is implicitly defined, any information generated by the resource lacks accountability. For this reason, the differences between a zeroth-order agent and a tool are indistinguishable.

A points profile agent is used to illustrate the zeroth-order agent. As shown in Figure 3, the points profile agent returns a set of normalized coordinate pairs representing a 2D unit circle (extension to circles of arbitrary diameter is simple).

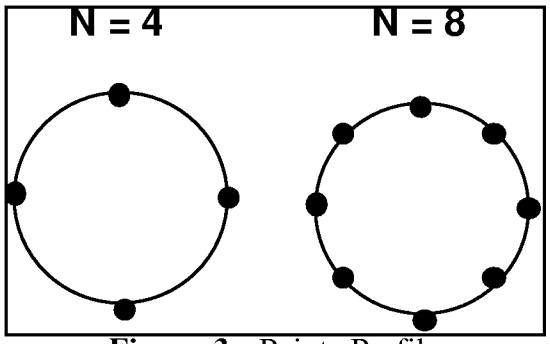


**Figure 3.** Points Profile

### Model

The points profile example uses a simple mathematical model, as shown in Figure 4. When queried for the circle description, the agent simply computes the normalized coordinate pairs based on the number of points needed and a uniform angular distribution.

### Resource

In this example, the resource implements the mathematical model in the C language, as seen in Figure 5. The program reads the number of profile points from the command line and writes the normalized coordinate pairs to standard output. Other implementations could be done in C++, FORTRAN, BASIC, PASCAL, UNIX shells, etc. The specific choice of implementation is not important as long as the resource acquires data from the command line and returns values through standard output.

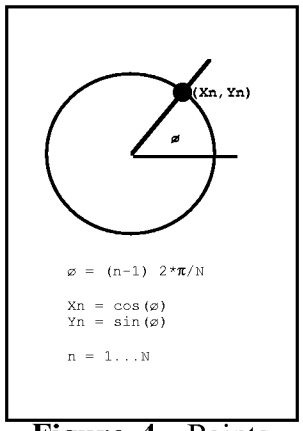


**Figure 4.** Points Profile - Model

```
/*
 *This program determines the points
 *profile for a unit circle.  N is
 *given on the command line.
 */
#include <stdio.h>
#include <math.h>
#define PI 3.141593
main (int argc,char *argv[])
{
   int N=atoi(argv[1]);
   int n;
   for ( n = 1; n <= N; n++)
      printf(" { %f %f } ",
         cos( (n-1)*2*PI/N),
         sin( (n-1)*2*PI/N));
}
```

**Figure 5.** Points Profile - Resource

## Wrap

```
#This procedure determines the points
#profile for a unit circle.  N is
#the number of points.
proc PointsProfile {N} {
   set C-Program [open "|circle $N" r]
   set Profile [gets $C-Program]
   return $Profile
}
```

**Figure 6.** Points Profile - Wrap

The wrap for the points profile example is done in Tk/tcl, as shown in Figure 6. When a request is made, the wrap forwards the number of points through a UNIX pipe to the command line of the resource. The coordinate pairs are then read from the standard output of the resource. Tk/tcl was used to wrap the resource since it provides agent communication and input/output stream utilities.

As shown in Figure 7, the points profile example is executed in the following manner:

1) The user or another agent queries the points profile agent wrap with the value "`PointsProfile 4`".
2) The wrap gives the resource the value `4` on the resource command line.
3) The resource calculates the normalized coordinate pairs based on the implicit mathematical model.
4) The coordinate pairs are returned to the wrap through standard output.
5) The wrap returns the coordinate pairs to the querying user or agent.

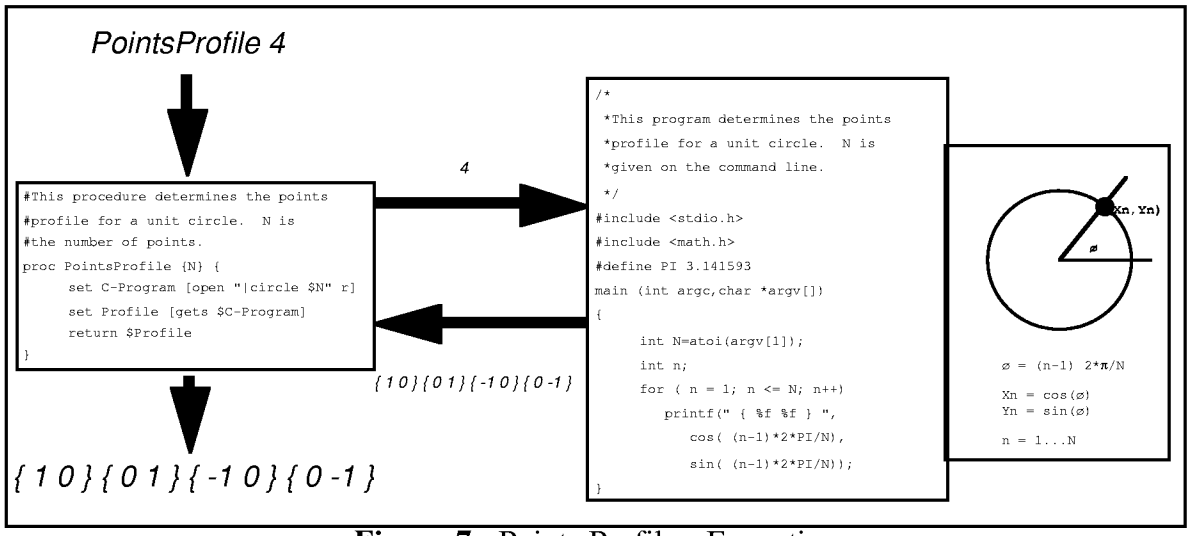


**Figure 7.** Points Profile - Execution

A number of important operating characteristics can be seen in this example:

1) The points profile wrap can be queried by any other agent or user on the network that understands the meaning of `PointsProfile`.
2) The resource can be changed without modifying the wrap and this can be done at run-time.
3) The model cannot be changed without modifying the resource.

## CATIA/FLOPS/ASTROS Visualization System

The IMAGE agent technology was used in another ASDL project to create a High Speed Civil Transport (HSCT) visualization system.[16] The system utilizes CATIA as a geometric modeling agent, FLOPS to create the HSCT geometry, and ASTROS to develop a wing structure finite element representation. A sample HSCT configuration is shown in Figure 8. A standard solid model of the HSCT and a wireframe representation of the wing structure FEM can be generated in approximately five minutes using this system. The HSCT visualization

system demonstrates the capability for integrating both proprietary and nonproprietary resources in an agent-based environment.

## CONCLUSIONS

The agent is one of the key technologies required for the implementation of integrated computing environments for Concurrent Engineering. The agent as defined in IMAGE has three basic components: the resource, the model, and the wrap. IMAGE agents formalize the role of proprietary resources in the implementation of computational design environments. Finally, IMAGE uses the agent to generate accountable information (information with context) that can be used throughout a product's life-cycle. A High Speed Civil Transport visualization system shows that the coupling of proprietary and nonproprietary agents is feasible. These examples illustrate the important role that agents can and will play in the design of the High Speed Civil Transport.

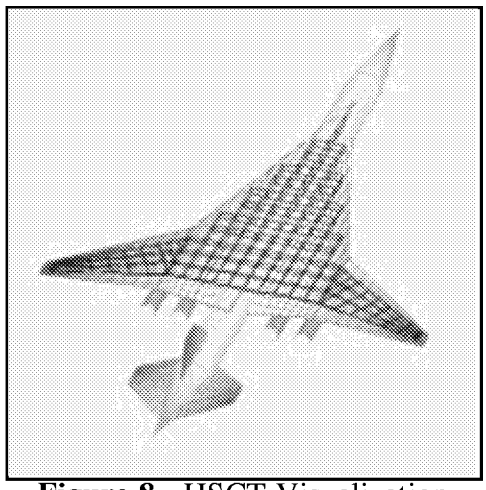

**Figure 8.** HSCT Visualization

## REFERENCES


1. Schrage, D.P. and J.E. Rogan, "The Impact of Concurrent Engineering in Aerospace Systems Design," course notes, 1991.
2. Townsend, J.C. and R.P. Weston, "An Overview of the Framework for Interdisciplinary Design Optimization (FIDO) Project," NASA TM 109058.
3. Jones, K.H. et al, "Information Management for a Large Multidisciplinary Project," AIAA-92-4720.
4. Dovi, A.R., et al, "Multidisciplinary Design Integration System For a Supersonic Transport Aircraft," AIAA-92-4841.
5. Cutkosky, M.R. et al, "PACT: An Experiment in Integrating Concurrent Engineering Systems," IEEE, Computer, January 1993.
6. Huyn, P.N., et al, "Automated Concurrent Engineering in Designworld," IEEE Computer, January 1993.
7. Hughes, D., "Generic Command Center Speeds Systems Design," Aviation Week & Space Technology, March 1993.
8. "ACSYNT Overview and Installation Manual," ACSYNT Institute, May 1992.
9. McCullers, L.A., "FLight OPtimization System, User's Guide," Version 5.41, NASA Langley Research Center, December 1993.
10. Kroo, I., et al. "A Quasi-Procedural, Knowledge-Based System for Aircraft Design," AIAA-88-4428.
11. Gage, P., et al, "Development of the Quasi-Procedural Method for Use in Aircraft Configuration Optimization," AIAA-92-4693.
12. Stephens, E.R. "LEGEND: Laboratory Environment for the Generation, Evaluation and Navigation of Design," Doctoral Dissertation, School of Aerospace Engineering, Georgia Institute of Technology, September 1993.
13. Hale, M.A. and J.I. Craig, "Preliminary Development of Agent Technologies for a Design Integration Framework," 5th AIAA/NASA/USAF/ISSMO Symposium on Multidisciplinary Analysis and Optimization, Panama City, FL, AIAA-94-4297.
14. Ousterholt, J.K., An Introduction to Tcl and Tk, Addison-Wesly Publishing Company, Inc: 1993.
15. Bhatia, K.G., and J. Wertheker, "Aeroelastic Challenges for a High Speed Civil Transport," AIAA-93-1478.
16. Marx, W.J., Schrage, D.P. and D.N. Mavris, "Integrated Product Development for the Wing Structural Design of the High Speed Civil Transport," 5th AIAA/NASA/USAF/ISSMO Symposium on Multidisciplinary Analysis and Optimization, Panama City, FL, AIAA-94-4253.